FAULT MANAGEMENT

OF WEB SERVICES

A Thesis Submitted to the College of

Graduate Studies and Research

In Partial Fulfillment of the Requirements

For the degree of Masters of Science

In the Department of Computer Science

University of Saskatchewan

Saskatoon

By

SAZEDUL ALAM

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Masters degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science

University of Saskatchewan

Saskatoon, Saskatchewan (S7N 5C9)

ACKNOWLEDGMENTS

I am grateful to my supervisor Dr. Ralph Deters for his guidance and advice. Without his guidance and support this work would not be completed. I thank him for everything he has done for me.

I also thank my advisory committee for their guidance and comments to make the thesis better.

I would like to thank my supervisor Dr. Ralph Deters, my department the Department of Computer Science and TRLabs for their funding in pursuing this graduate program.

Finally, I would like to thank my family and friends for their love and support.

ABSTRACT

The use of service-oriented (SO) distributed systems is increasing. Within service orientation web services (WS) are the de facto standard for implementing service-oriented systems. The consumers of WS want to get uninterrupted and reliable service from the service providers. But WS providers cannot always provide services in the expected level due to faults and failures in the system. As a result the fault management of these systems is becoming crucial. This work presents a distributed event-driven architecture for fault management of Web Services. According to the architecture the managed WS report different events to the event databases. From event databases these events are sent to the event processors. The event processors are distributed over the network. They process the events, detect fault scenarios in the event stream and manage faults in the WS.

TABLE OF CONTENTS

page

LIST OF TABLES

Table page

LIST OF FIGURES

Figure page

LIST OF ABBREVIATIONS

BPEL - Business Process Execution Language

CIM - Common Information Model

ED - Event Database

EFG - Event File Generator

EOG - Event Object Generator

EP - Event Processor

ER - Event Router

ESB - Enterprise Service Bus

GFM - Global Failure Manager

HTTP - Hyper Text Transfer Protocol

J2EE - Java 2 Platform, Enterprise Edition

JAXB - Java Architecture for XML Binding

JAX-WS - Java API for XML Web Services

JDBC - Java Database Connectivity

JMS - Java Message Service

JMX - Java Management eXtensions

MUWS - Management Using Web Services

MOWS - Management of Web Services

OASIS - Organization for the Advancement of Structured Information Standards

PO - Purchase Order

POMWS - Purchase Order Management Web Service

QoS - Quality of Service

REST - Representational State Transfer

RMI - Remote Method Invocation

SAAJ - SOAP with Attachments API for Java

SLA - Service Level Agreement

SMTP – Simple Mail Transfer Protocol

SNMP - Simple Network Management Protocol

SO - Service Orientation

SOA - Service Oriented Architecture

SOAP - Simple Object Access Protocol

SOC - Service-Oriented Computing

UDDI - Universal Description Discovery and Integration

URL - Uniform Resource Locator

WS - Web Services

WSDM - Web Services Distributed Management

WSMS - Web Services Management System

W3C - World Wide Web Consortium

WSDL - Web Services Definition Language

XLANG - XML-based extension of Web Services Description Language

XML – eXtensible Markup Language

CHAPTER 1
INTRODUCTION

In the last five years service-orientation (SO) has emerged as a new system design/integration paradigm for building distributed systems. The Service-Oriented-Architecture (SOA) is a paradigm for designing a software system that offers services to either end-user applications or to other services. SOA was first introduced by Gartner in April 1996. As a system design/integration philosophy it is independent of specific technologies, e.g., Web Services (WS) or J2EE [1].

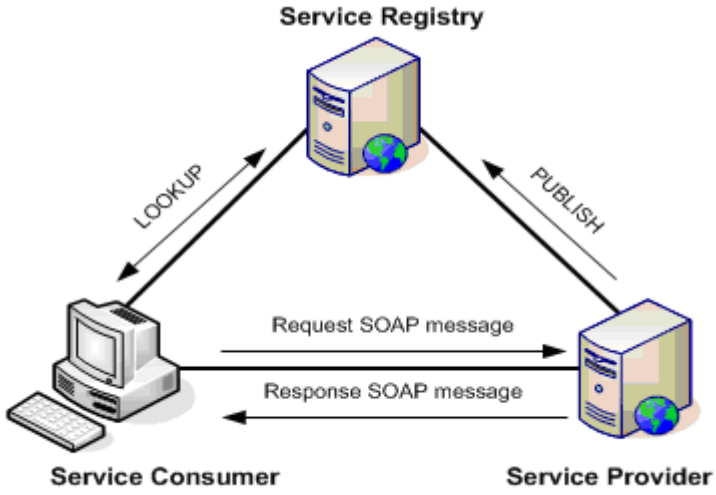

Figure 1-1. Service Oriented Architecture

The main building blocks of SOA are – service provider, the service registry, and the service consumer. Service providers are software applications that provide the service to the service consumer. The provider publishes a description of the services they provide via a service registry. Service consumers consume the service. Any software application can be both service client and service provider at the same time. Clients must be able to find the description of the services they require and must be able to bind to them.

Web services (WS) are the de facto standard for implementing service-oriented systems. WS enable different applications running on different machines to exchange data and integrate with one another without requiring additional software. The applications that are built on WS technology can exchange data regardless of the language, platform, or internal protocols they use.

The WS providing organization owns the WS and implements business logic that underlies the service. WS are published in a service registry which is hosted by a service discovery agency. The description of the WS contains information on the business, service, and technical information of the WS. The WS publisher has to describe this information with a registry in a predefined format specified by the discovery agency.

Currently there are two different ways to develop WS. One is SOAP (Simple Object Access Protocol) based WS and the other is RESTful WS. REST stands for Representational State Transfer. SOAP was developed at Microsoft and was designed to be platform and language independent. It has become the standard for exchanging XML-based messages. Conceptually SOAP is more difficult than REST. REST assumes a point to point communication model and is tied to the HTTP transport model. The use of SOAP based WS and RESTful WS both are popular with service providers.

An example of SOAP request and response message, taken from [45], is shown here. In this example a GetStockPrice request is sent to a server which takes StockName as the parameter and the Price will be returned in the response from the server.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>

</soap:Envelope>
```

Figure 1-2: A SOAP request message [45]

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>

</soap:Envelope>
```

Figure 1-3: A SOAP response message [45]

In RESTful WS the request may be any url `http://www.example.org/instock/ibm` and the HTTP request will be a simple GET request as shown below.

```
GET /instock/ibm HTTP/1.1
User-Agent: SomeClient
Host: example.org
Accept: text/html
```

Figure 1-4: A Simple RESTful WS request

Similarly the response may be something like as shown in the following figure.

```
HTTP/1.1·200·OK
Server:·Apache
Connection:·close
Content-Type:·text/html

<?xml version="1.0"?>
<GetStockPriceResponse>
<Price>
        34.5
</Price>
</GetStockPriceResponse>
```

Figure 1-5: A Simple RESTful WS response

As a WS network grows, its existence and performance becomes crucial to the business's core activities. So the management of WS is important for providing seamless access of the service to the user. The important functions of service management include Service Level Agreement (SLA) management, auditing, monitoring, troubleshooting, service redeployment, dynamic routing and graceful degradation, service life cycle and state management, dynamic service provisioning, security management, service maintenance, and service management for WS infrastructure and applications [1].

This research focuses on the fault management of SOAP based WS due to their wide acceptance in the WS provider community. This work focuses on services where multi-faults can occur simultaneously. The main goal of this research is to detect and manage faults in SOAP based WS.

Here I propose an architecture of a system for fault management of WS. The proposed system is based on distributed event processing. The main components of the system are enterprise service bus, event database, event file generator, event object generator, event router and event processor. All components of the proposed architecture are distributed over the network. According to my system, all the WS will generate different events in its lifetime. These

events will be reported and stored in the event databases. I assume that there will be no noise in reporting of the events and no event will be lost. I also assume that the event databases are a reliable storage of events. The event processors will get events from the event databases and they will process the events to detect faults.

The rest of the paper is organized as follows. Chapter 2 describes the problem definition. Chapter 3 presents related terms and related works on web services management. Chapter 4 presents the proposed architecture. Chapter 5 presents the experimental setup and results. Chapter 6 presents conclusion and expected contribution.

CHAPTER 2
PROBLEM DEFINITION

The use of WS applications is increasing daily. Their size and complexity is also increasing to meet functional requirements. WS can be found in Online Transaction Processing applications, Banking applications, Database Management applications, Groupware applications, legacy resources and so on. As a result, we become increasingly dependent on them - making it necessary to monitor, track and manage them.

The consumers of WS want to get uninterrupted and reliable service from the service providers. But WS providers cannot always provide services to the expected level due to faults and failures. WS fail to perform in accordance with the service level objectives. So services need to be monitored for availability and performance from the user's perspective [1].

The faults can occur in different places, for example faults can occur in the software application, in the network connection, or in the hardware resources. So the resources that are distributed over the network have to be monitored for availability, performance, and utilization.

Traditional fault management tools can't automatically monitor, analyze, and resolve faults in WS when a large number of services, suppliers and technologies collaborate together to perform a business level activity. As WS are distributed in nature, it is hard to trace the flow of transactions within a distributed system. This makes monitoring and fault management of WS more difficult than centralized applications [1].

Let us consider a purchase order management web service (POMWS). The POMWS manages purchase orders submitted to a specific supplier by a client. There are some steps involved before the client gets an invoice for the purchase order (PO) from POMWS. Then the client confirms the PO. This completes the ordering process of the client.

For simplicity, let us assume that the following steps are performed by POMWS when a client submits a PO.

1. POMWS calls credit checking WS to check the credit worthiness of the client.
2. POMWS calls the supplier's inventory WS to check if the items are available.
3. POMWS calls shipping WS to calculate the shipping cost.
4. POMWS calls billing WS to calculate the total bill
5. POMWS sends the invoice to the client.

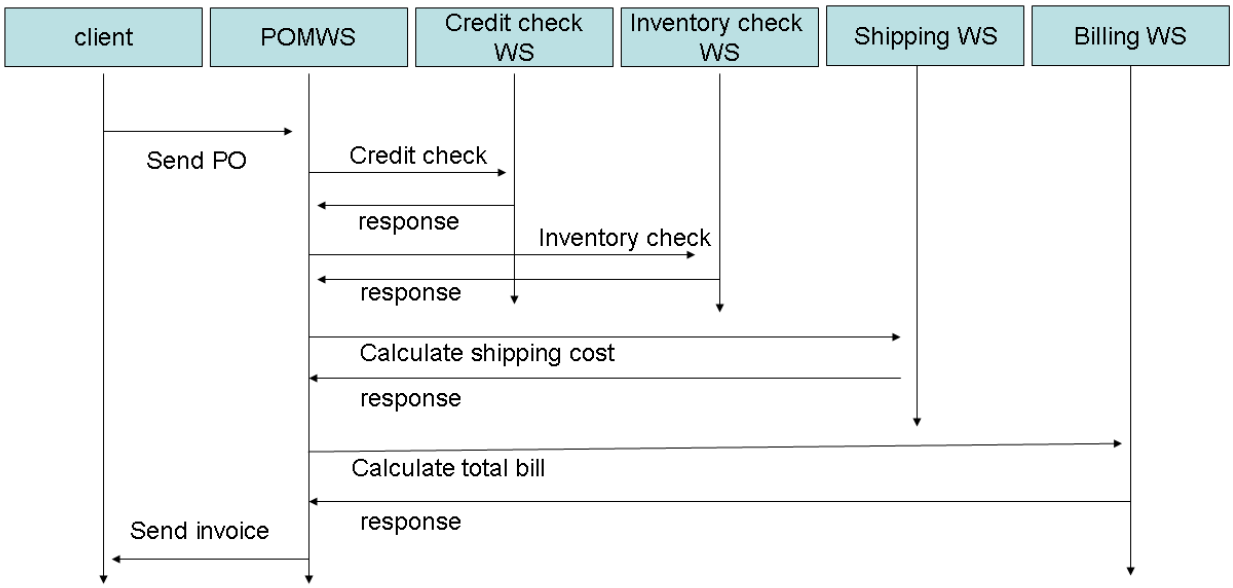

Figure 2-1: Simple workflow of POMWS

During a purchase order management different faults can occur and they can cause failure to complete the order. For example any of the participating WS can be unavailable to provide its service. The WS can be down because of a hardware fault or software fault, or the network connection can be down between POMWS and a specific WS. A PO can fail also due to invalid data in it. For example, the customer provides an invalid postal code and the shipping WS

identifies the invalid code when it tries to find the shipping cost. This fault causes failure to complete the order.

If credit checking WS, inventory WS, shipping WS and billing WS call other different services to complete their services then a series of nested service calls is involved to complete a PO. It is very hard to track and manage faults in this type of scenario. If POMWS has a fault management system which does not know about the services at the nested levels, it will not be able to find the root cause of the fault. So tracking and managing faults in a distributed environment need a new approach.

In this research I want to investigate the following questions:

1. How can we monitor SOAP based WS for faults?
2. How can we detect faults in SOAP based WS?
3. Which faults can be detected? Can the system detect faults if multiple faults occur simultaneously?
4. What is the latency between occurrence and detection of faults? How can we improve it?
5. How can WS recover when any fault occurs?

CHAPTER 3
RELATED LITERATURE

This chapter presents related technical terms and related works on management of distributed systems and management of web services. The works on management of distributed systems are related to fault management of WS as WS are distributed in the network.

3.1 Web Services (WS)

WS are self-describing, self-contained software modules available via a network, such as the internet, which completes tasks, solves problems, or conducts transactions on behalf of a user or application. WS constitute a distributed computer infrastructure made up of many different interacting application modules trying to communicate over a private or public network to virtually form a single logical system [1].

WS are loosely coupled software modules. The service interface is defined in a neutral manner that is independent of the underlying platform, the operating system, and the programming language the service is implemented in. This allows services, built on a variety of such systems, to interact with each other in a uniform and universal manner.

A WS is a self contained software module that performs a single task. The modules describes its own interface characteristics, i.e. the operations available, the parameters, data typing, and the access protocols, in such a way that other software modules can invoke its functionality and know what results to expect.

WS can be accessed programmatically. A web service provides programmable access – this allows embedding WS into remotely located applications.

WS can be dynamically found and included in applications at compile or runtime. WS are described in terms of a standard description language. The Web Services Description Language,

WSDL, describes both functional as well as non-functional service characteristics. Functional characteristics include operational characteristics that define the overall behavior of the service while non-functional characteristics mainly describe characteristics of the hosting environment.

WS are distributed over the internet and make use of existing, ubiquitous transport internet protocols like HTTP. This helps to comply with current corporate firewall policies.

3.1.1 Types of WS

There are two types of web services – Simple or Informational services and Complex services or Business processes.

There are two types of WS based on the protocol they use – SOAP based WS and REST (Representational State Transfer) WS. In REST WS the web services are viewed as resources that can be identified by their URLs [43].

3.1.1.1 Simple or informational services

Informational services are of a simple nature. They provide access to content interacting with an end user by means of simple request/response sequences, or alternatively may expose back-end business applications to other applications. Informational services can be divided into three subcategories according to the business problem they solve. These are pure content services, simple trading services, and information syndication services.

3.1.1.2 Complex services or business process

Complex or composite services typically involve the assembly and invocation of many pre-existing services found in diverse enterprises to complete a multi-step business interaction.

Complex services can be divided into two groups – complex services that compose programmatic WS and complex services that compose interactive WS.

3.1.2 Web Services Technology Stack

WS allow communication between applications regardless of their platforms, languages, and operating systems. So the WS technology has to allow applications to work together over standard internet protocols, without direct human intervention.

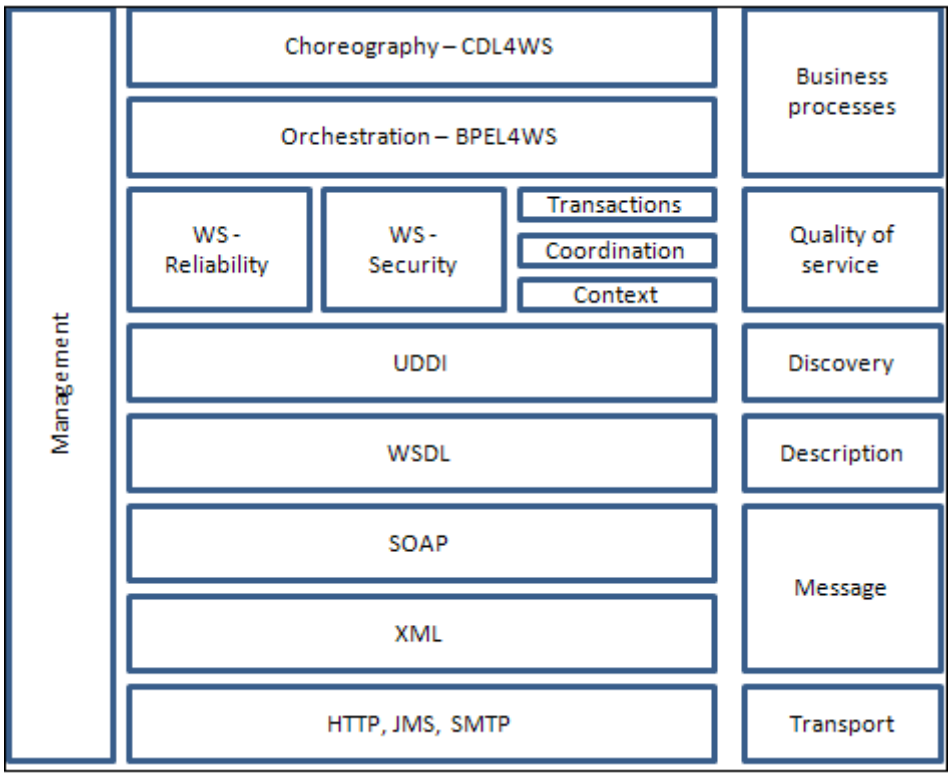

Figure 3-1: WS technology stack [1]

WS represent a collection of several related technologies. As a result the WS technology stack consists of many collaborating standards.

3.1.2.1 Enabling technology standards

At the transport level, WS use HTTP, or JMS, SMTP. WS use XML as the fundamental building block for nearly every other layer in the WS stack.

3.1.2.2 Core services standards

At the core the baseline standards are SOAP, WSDL, and UDDI.

3.1.2.3 Service composition standards

Service composition describes the execution logic of WS-based applications by defining their control flow (such as conditional, sequential, parallel, and exceptional execution) and prescribing the rules for consistently managing their unobservable data.
The Business Process Execution Language (BPEL) can achieve composition for WS [4].

3.1.2.4 Service collaboration standards

Service collaboration describes cross-enterprise collaborations of web service participants by defining their common observable behavior, where synchronized information exchanges occur through their shared contact points, when commonly defined ordering rules are satisfied. Service collaboration is materialized by the web services choreography description language WS-CDL [7], which specifies the common observable behavior of all participants engaged in business collaboration.

3.1.2.5 Coordination/transaction standards

Coordination/transaction standards provide mechanisms for defining specific standard protocols for use by transaction processing systems, workflow systems, or other applications that wish to coordinate multiple WS.

3.1.2.6 Value added standards

Value added standards include mechanisms for security and authentication, authorization, trust, privacy, secure conversations, contract management etc.

3.2 Simple Object Access Protocol (SOAP)

SOAP is an XML-based communication protocol for exchanging messages. WS rely on SOAP for exchanging messages between computers regardless of their operating systems, programming environment, or object model framework.

A SOAP XML document instance is called a SOAP message or SOAP envelope. It is carried as the payload of some other network protocol like HTTP. The SOAP message consists of an <Envelope> element containing an optional <Header> and a mandatory <Body> element [3].

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2002/12/soap-envelope">

<env:Header>
   .......
   .......
</env:Header>

<env:Body>
   .......
   .......
 </env:Body>

</env:Envelope>
```

Figure 3-2: Construct of a SOAP message

SOAP has a clear purpose: exchanging data over networks. It concerns itself with encapsulating and encoding XML data and defining the rules for transmitting and receiving that data [44].

3.3 Web Services Description Language (WSDL)

WSDL is an XML-based specification schema for describing the public interface of a web service. This public interface can include operational information relating to a web service such as all publicly available operations, the XML message protocols supported by web service, data type information for messages, binding information about the specific transport protocol to be used, and address information for locating the web service.

WSDL defines six major elements. They are:

1 Types: types provide data type definitions used to describe the messages exchanged.

2 Message: message represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.

3 Port Type: port Type is a set of abstract operations. Each operation refers to an input message and output messages.

4 Binding: binding specifies the concrete protocol and data format specifications for the operations and messages defined by a particular port Type.

5 Port: port specifies an address for a binding, thus defining a single communication endpoint.

6 Service: service is used to aggregate a set of related ports.

3.4 Business Process Execution Language (BPEL)

BPEL now is an OASIS standard for defining workflows. Using BPEL web services can be orchestrated. BPEL mainly focuses on modern business processes.

BPEL distinguishes five main sections: the message flow, the control flow, the data flow, the process orchestration, and the fault and exception handling sections.

Workflows are related to business processes. A workflow system automates a business, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules [6].

3.5 Enterprise Service Bus (ESB)

The Enterprise Service Bus is an open standards-based backbone designed to enable the implementation, deployment, and management of SOA-based solutions with a focus on assembling, deploying, and managing distributed service oriented architectures. An ESB is a set of infrastructure capabilities implemented by middleware technology that support the SOA and alleviate disparity problems between applications running on heterogeneous platforms and using diverse data formats. The key capabilities of an ESB are listed below [13] [14] [15] [16].

1 Dynamic connectivity capability:

2 Reliable messaging capability.

3 Topic and content based routing capability.

4 Transformation capability.

5 Service enablement capability.

6 Endpoint discovery with multiple QoS capability.

7 Long running process and transaction capability.

8 Security capability

9 Integration capability.

10 Management and monitoring capability.

11 Scalability capability.

Implementing an ESB requires an integrated set of middleware facilities that support the following interrelated architectural styles [17].

1 In the SO architecture distributed applications are composed of granular reusable services with well-defined, published, and standards-compliant interfaces.

2 In Message-driven architectures applications send messages through the ESB to receiving applications.

3 In Event-driven architectures applications generate and consume messages independently of one another.

3.6 Event

"An event is an object that is a record of an activity in a system. The event signifies the activity". An event has three aspects - form, significance and relativity [54]. The event's form is

an object which can be a single value, or a set of data components. The following figure shows an event class in Java.

```
public class Event implements Serializable
{
        private int eventId;
        private String eventSource;
        private String reporterId;
        private int eventPriority;
        private long reportTime;
        private String eventDescription;

...
}
```

Figure 3-3: Form of an event as a Java class

"An event signifies an activity" [54]. From the values of the data components we get the description of the activity. "The relationships between an event and other events are together called its relativity." An event is often related to other events by time, causality and aggregation [54]. If an event is created when a set of event happens then the created event is the aggregation of the set of the happened events. For example if event A is created for a set of events $\{B_i\}$ then A is the aggregation of the events of $\{B^i\}$. A is also a complex event.

3.7 Web Services Eventing (WS-Eventing)

When events occur in different services and applications some WS want to receive messages about the event (subscribe to the event). For receiving events the WS has to subscribe to the event source. WS-Eventing specification defines protocol for the subscriber WS to subscribe with event source. [23]

This specification intends to meet the following requirements:

1. Define means to create and delete event subscriptions.
2. Define expiration for subscriptions and allow them to be renewed.

3. Define how one Web service can subscribe on behalf of another.
4. Define how an event source delegates subscription management to another Web service.
5. Allow subscribers to specify how event messages should be delivered.
6. Leverage other Web service specifications for secure, reliable, transacted message delivery.
7. Support complex eventing topologies that allow the originating event source and the final event sink to be decoupled.
8. Provide extensibility for more sophisticated and/or currently unanticipated subscription scenarios.
9. Support a variety of encoding formats, including (but not limited to) both SOAP 1.1 and SOAP 1.2 Envelopes.

3.8 Case Based Reasoning

Case-based reasoning (CBR) is a major paradigm in automated reasoning and machine learning. In CBR, a new problem is solved by finding its similarity to one or several problems solved previously and by adapting their known solutions instead of working out a solution from scratch [24]. So in CBR, reasoning is done by remembering the previous solutions and the case based reasoner learns from its experience [31].

The field of CBR was first introduced by Roger Schank and his colleagues [25] [26]. Cognitive Science and Artificial Intelligence are the main two fields that used CBR. Cognitive science tried to model human behavior using CBR. Some pioneering works in this field are done by Ross [32][35], Pirolli & Anderson [33], Faries & Schlossberg [34], Lancaster & Kolodner [35], Schmidt, Norman, & Boshuizen [36], Read & Cesa [37], Klein & Calderwood [38][39].

Other pioneering works in CBR are done by Carbonell on Analogy [28], Kolodner on reconstructive memory [29] and Rissland on legal reasoning [30].

In AI technology CBR can be helpful in five fields [31]. They are

1. Knowledge acquisition
2. knowledge maintenance
3. increasing problem solving efficiency
4. increasing quality of solutions
5. user acceptance

There are two types of CBR tasks [40] [42]. They are interpretive CBR and problem solving CBR.

Given a case to solve, interpretive case-based reasoning involves the following steps:

1. Retrieving relevant cases from the case memory (this requires indexing the cases by appropriate features); this step is called situation assessment in [40] [41].
2. Selecting a set of best cases;
3. Deriving a solution; evaluating the solution (in order to make sure that poor solutions are not repeated);
4. Storing the newly solved case in the case memory.

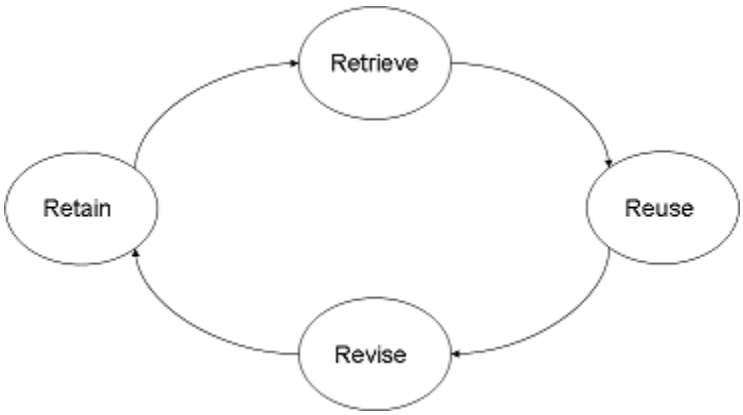

Figure 3-4: The 4 R's in CBR

According to these steps, Aamodt and Plaza describe a Case-Based reasoner as a cyclic process comprising "the 4 R's" i.e. Retrieve, Reuse, Revise and Retain [27].

Like interpretive CBR, problem-solving CBR performs situation assessment, case retrieval, and similarity assessment/evaluation. Problem solving CBR also finds the similarities and differences between new and prior cases. Then these similarities and differences are used to determine how the solution of the previous case can be adapted to solve the new situation.

3.9 Fault and Failure

IEEE Standard Glossary of Software Engineering Terminology defined fault and failure [55]. According to IEEE a fault is (1) A defect in a hardware device or component; for example, a short circuit or broken wire. (2) An incorrect step, process, or data definition in a computer program.

According to IEEE, "failure" is the inability of a system or component to perform its required functions within specified performance requirements. The fault tolerance discipline

distinguishes between a human action (a mistake), its manifestation (a hardware or software fault), the result of the fault (a failure), and the amount by which the result is incorrect (the error).

In a nutshell, a fault is the cause of failure and failure is the result of fault.

3.10 Terms Related to Management of Distributed Systems

In [1] Michael P. Papazoglou defines “A managed or manageable resource in a distributed environment could be any type of a hardware or software component that can be managed and that can maintain embedded management related metadata. A managed resource could be a server, storage unit, database, application server, service, application, or any other entity that needs to be managed.” A distributed application management system controls and monitors an application throughout its lifecycle - from installing and configuring to collecting metrics and tuning the application to ensure responsive execution [47].

In [9] Justin Murray emphasizes designing and developing software applications that are readily manageable. He defines "Manageability" as the ability to exercise administrative and supervisory actions and receive information that is relevant to such actions, on a component. He also states that manageability requires a mechanism to ensure that an application is both active and functioning properly, as well as the ability of checking an application’s performance over time. According to him “Manageability” can be distinguished by three functional parts:

1. Monitoring: The ability to capture run-time and historical events from a particular component, for reporting and notification.

2. Tracking: the ability to observe aspects of a single unit of work or thread of execution across multiple resources.
3. Control: The ability to alter the run-time behavior of a managed resource.

M. P. Papazoglou also emphasizes monitoring in [1]. He proposes that distributed management solutions should have components for user experience monitoring, infrastructure monitoring, transaction monitoring, resource provisioning, SLA monitoring.

According to T. Mehta any management solution must address four necessary management questions [10].

1. What resources need to be managed?
2. What are their properties?
3. How the management information is exchanged (operations, notifications, and kinds of protocols needed)?
4. What are the relationships among the managed resources?

In [11] D. Kakadia identifies enterprise management systems as network management systems capable of managing devices, independent of vendors and protocols, in IP based enterprise networks. Most Enterprise management systems use functions like fault management, configuration management, accounting management, performance management, and security management.

3.11 Protocols and Frameworks for Management of Networked Systems

There are many protocols and frameworks already in use for management of computer systems in the network.

The SNMP (Simple Network Management Protocol) is an application layer protocol for network management [22]. The SNMP framework lets network administrators manage network performance, find and solve network problems, and plan for network growth.

The Common Information Model (CIM)/ Web based Enterprise Management also provides a solution for management of elements across the enterprises, including systems, networks, and applications [21].

Java Management Extensions (JMX) technology provides the tools for building distributed, web-based, modular, and dynamic solutions for managing and monitoring devices, applications, and service driven networks [18].

With SOA, a standard WS management framework can provide support for discovering, introspecting, securing, and invoking managed resources, management functions, and management infrastructure services and toolsets [1]. A WS management framework is a set of components and objects that enable SLA management, auditing, monitoring, troubleshooting, dynamic service provisioning, and service management for WS infrastructure and applications. OASIS is developing Web Services Distributed Management (WSDM) standards to provide end-to-end WS management. WSDM developed specifications on two standards - Management Using Web Services (MUWS) [19] and Management of Web Services (MOWS) [20].

Management Using Web Services (MUWS) enables management of WS using WS. MUWS provides management capability for monitoring the quality of a service, enforcing a service level agreement, controlling a task, and managing a resource lifecycle.

MUWS is able to work with multiple, existing, domain-specific models. CIM from DTMF is the most prominent model for this work. SNMP information model is also considered. According to MUWS access to the manageability for a resource is provided by a web service

endpoint. The web service endpoint providing access to some managed resource is called a manageability endpoint. A manageability endpoint realizes a number of management interfaces. Each management interface represents all or part of a manageability capability. Similarly, a single manageability capability may be represented in one or more interfaces.

Management of Web Services (MOWS) [20] is based on the concepts and definitions expressed in the MUWS specification. WSDM MOWS defines the MUWS capabilities that are applicable to WS endpoints are - Identity, Identification, Metrics (Number of requests, number of failed requests, number of successful requests, service time, max response time and last response time), Operational State, Operational status, request processing state.

3.12 Related Works on Fault Management of WS

Fault management of WS is a relatively a new field in research. Not much work has been done on this topic. Most of the works presented by the researchers are in ongoing stage. Most of the papers on this topic do not show any experimental results. The papers mostly present the architecture of the proposed systems.

D. Ardanga et al. presents a classification of faults in web services and a self-healing platform for recovery of faults [48]. The paper classifies WS faults into three levels – infrastructural and middleware level, web service level, and web application level. This paper presents a self-healing platform which implements the run time service oriented fault analysis and recovery actions. The following table shows the fault types as presented by D. Ardanga et al.

Table 3-1. Fault types in [48].

Level	Fault type	Examples
Web Applications	1. internal data fault	1. data quality faults – value mismatch, missing data.
	2. application coordination faults	2. resource unavailable at runtime
	3. actor faults	3. customer not connected when synchronous communication is needed.
	4. QoS violation faults	4. QoS value beyond threshold
Web Service	1. WS execution faults	1. missing part in input message
	2. WS coordination faults	2. process failure or timeout
Infrastructural & middleware	1. node faults	1. node (application server or client device) down
	2. network faults	2. missing connection, low bandwidth
	3. generic faults	3. Denial of service, wrong authentication

For correct recovery actions the system in Ardanga et al. has a log of faults for different levels. The fields of a log contain a fault id, a fault type id and a time stamp. The web application level log contains more specific fault parameters and a pointer to the logs of lower level faults that originated the fault. The recovery actions in Ardanga et al. are shown in the following table.

Table 3-2. Recovery actions in [48].

Recovery action type	Actions	Fault type	Type
Service oriented recovery action	Retry	Infrastructural, WS execution	Reactive
	Substitute	WS execution and coordination	Reactive
	Completion of missing message parts	WS execution	Reactive
	Reallocate	QoS	Reactive/proactive
	Change process structure	All	Proactive
	Process oriented methods	Application level	Proactive
Data quality recovery actions	Insert data quality block	Internal data	Reactive
	Process oriented methods	Application level	Proactive

The architecture of the system has four modules to handle recovery action. They are

1. reallocation module
2. substitution module
3. wrapper generator module
4. quality module

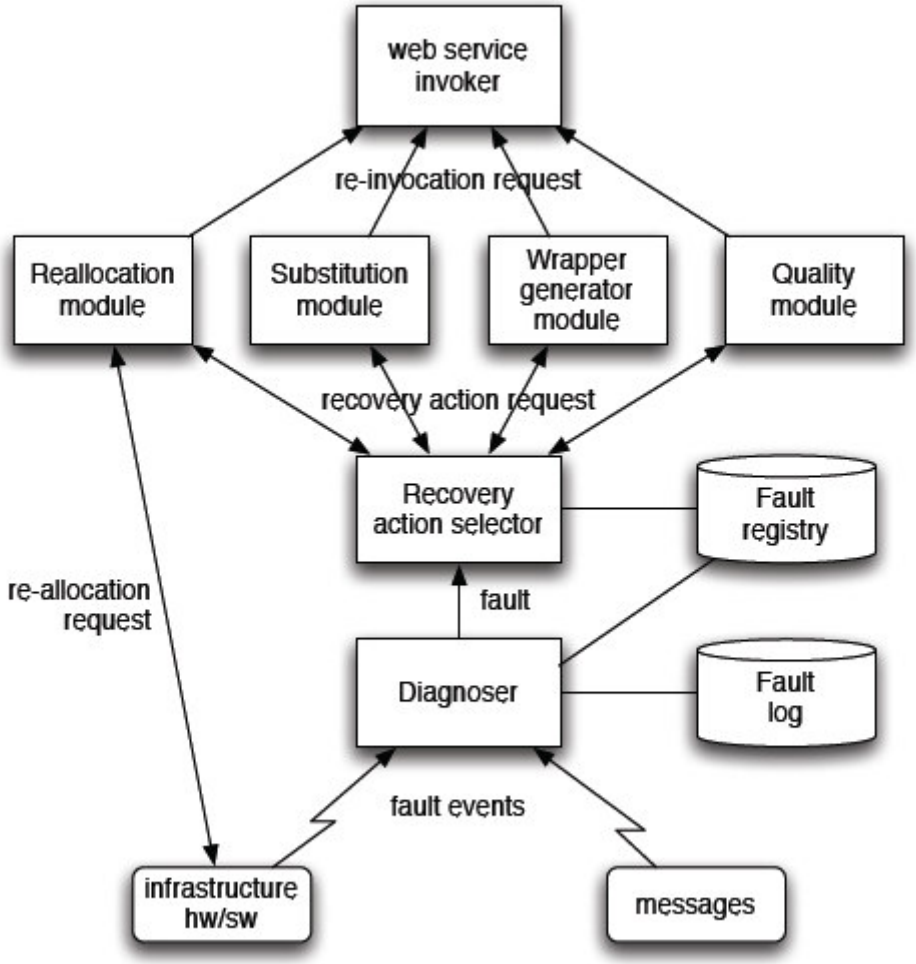

Figure 3-5: Self-healing platform architecture [48]

The paper shows the theoretical representation of the system [48]. It does not present any experimental data.

He proposes an infrastructure to implement failure recovery capabilities in WS management systems [49]. He classifies failures in a Web Services Management System (WSMS) into five categories - functional failures, operational failures, semantic failures, privacy failures, and security failures.

Table 3-3. Failure types in [49].

Failure type	Cause of failure	Example
Functional failure	Software bug, hardware faults in the system	Computer hard drive crashed.
Operational failure	System or some participant service is unavailable due to communication problem or unpredictable load.	Heavy load in an air ticket booking WS makes it unable to accept new requests.
Semantic failure	Interacting operations between two participants are not compatible due to different ontology. Message exchanged between two services are incompatible.	Hotel reservation WS and car rental WS communicates between each other and does not have same time format.
Privacy failure	Service or data are inaccessible because they are privacy sensitive and WS would not disclose information to everybody.	Hotel reservation WS may not provide age of the customer to Car rental WS.
Security failure	Data are accessed without enough credential or authority, or without special secure link.	the car rental Web Service only accepts SOAP messages over the secure HTTP, while the WSMS sends SOAP messages over standard HTTP

The proposed architecture is shown in the following figure.

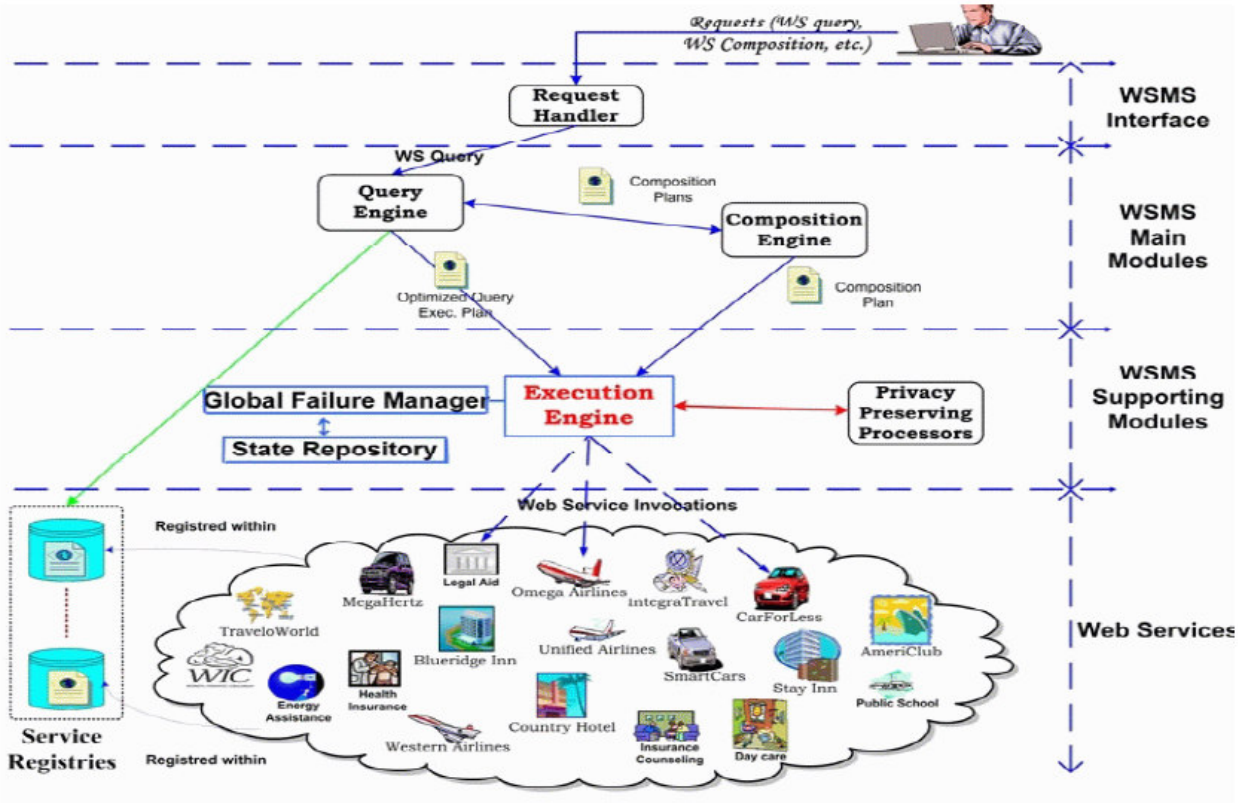

Figure 3-6: Reliable WS application infrastructure [49]

In this work users' requests are received by the Request Handler. The Request Handler interprets requests and checks their syntactic and semantic validity. Then a composite service is dynamically generated through the Composition Engine. The queries over Web services are processed and optimized by the Query Engine. Optimized query execution and composition plans are executed by the Execution Engine. The Privacy Preserving Processor ensures that all privacy requirements, from users, Web services, and data perspectives, are fulfilled. The Global Failure Manager (GFM) keeps a trace of all executions and repairs the system in case of failures. GFM monitors faults, conducts failure diagnosis and recovery coordination. The components of GFM are shown in the following figure.

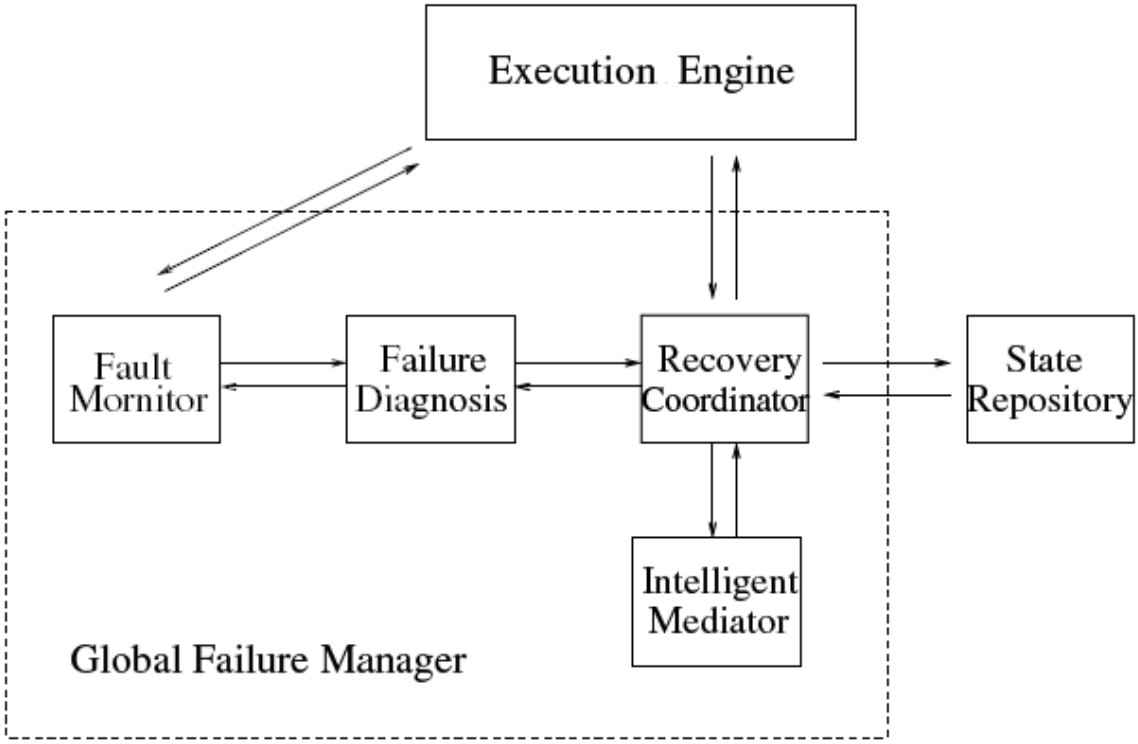

Figure 3-7: Global Failure Manager [49]

He indicates that the work is ongoing. However there is no experimental data.

Benharref et al. presents a web service based architecture for detecting faults in web services [51]. The main component of the architecture is an observer. Observers are WS and published through UDDI. The observer is invoked by a manager. When the observer finds faults or the observation period expires it returns the result to the manager. The following figure shows the steps involved in reporting faults.

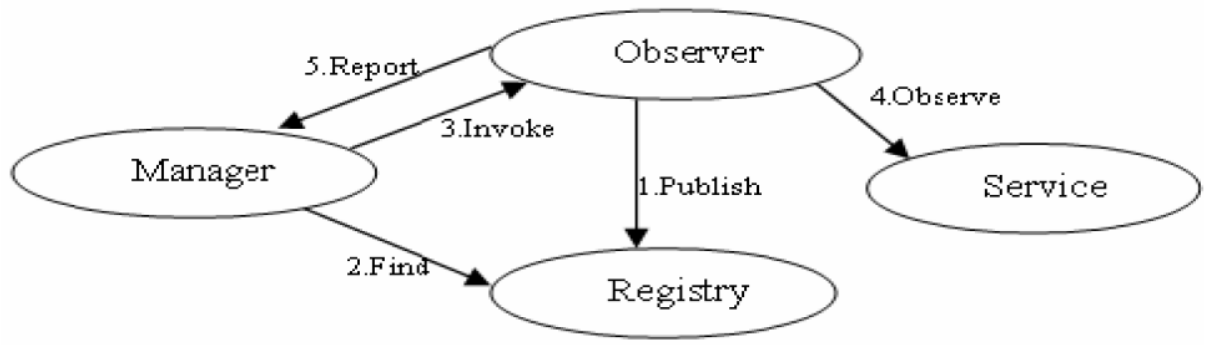

Figure 3-8: Fault detection and reporting step [51]

In the paper, experimental results were not shown. In the presence of deadlocks the system does not work.

Masoud Mansouri et al. present a functional model for monitoring of distributed systems [50]. This monitoring model has a set of monitoring functions: generation of monitoring information, processing of the information, dissemination of the information, and presentation of the information. The work presents a monitoring model only. It does not include overall management solution. It also does not present any results.

Ertugrul Akbas presents an RMI and JAVA based architecture of system independent and distributed fault management system [52]. However his work also does not show any results.

Pankaj Kumar, HP presents best practices that can be used by web services during distributed system management [53]. These best practices are defined in the form of principles. They are : Service orientation has to be properly thought through, Design for evolution, Write wrappers (rather than doing a complete reimplementation), Handle failure conditions gracefully, Create models, Realize that you are an early adopter of technology. This paper has only a theoretical presentation of the system. No quantitative result is shown.

Table 3-4: Summary of Related Works

Reference Paper	Work and contribution	Limitations
D. Ardanga et al. [48]	Presents classification of faults in web services and a self-healing platform for recovery of faults. This paper classifies WS faults into three levels – infrastructural and middleware level, web service level, and web application level. This work presents a self-healing platform which implements the run time service oriented fault analysis and recovery actions.	This paper shows the theoretical representation of the system. It does not present any experimental data.
Weiping He [49]	Proposes an infrastructure to implement failure recovery capabilities in WS management systems. He classifies failures in WSMS into five categories - functional failures, operational failures, semantic failures, privacy failures, and security failures.	Does not show any experimental data.
Masoud Mansouri et al. [50]	Presents a functional model for monitoring of distributed systems. Presented a monitoring model that has a set of monitoring functions: generation of monitoring information, processing of the information, dissemination of the information, and presentation of the information.	Presents monitoring model only. Does not include overall management solution. It also does not present any results.
A. Benharref et al. [51]	Presents a web service based architecture for detecting faults in web services.	Results were not shown. For deadlocks the system does not work.
Ertugrul Akbas [52]	Presents RMI and JAVA based architecture of system independent and distributed fault management system.	This one also does not show any result.
Pankaj Kumar, HP [53]	This paper presents some best practices that can be used by web services during distributed system management.	Theoretical presentation. No quantitative result is shown.

All of the papers cited in the related works lack quantitative results. The authors did not show any experimental data. So we do not know how the systems perform. There are also some additional open questions regarding the proposed systems.

1. How long does it take to detect and manage faults?
2. Can it detect all the faults?

3. Is the system scalable?
4. Is the system reliable?

CHAPTER 4
PROPOSED ARCHITECTURE

In this research work I introduce a case based distributed event processing tool for fault management of WS. It has the following major components:

1 Enterprise Service Bus (ESB)

2 Event Database (ED)

3 Event File Generator (EFG)

4 Event Object Generator(EOG)

5 Event Router (ER)

6 Event Processor (EP)

In the proposed architecture I made the following assumptions

1. Multiple faults can occur in the system simultaneously.
2. Databases in the system are reliable and available all the time.
3. There will be no noise in the case of reporting the events i.e. all the events will be reported and no event will be lost or garbled.

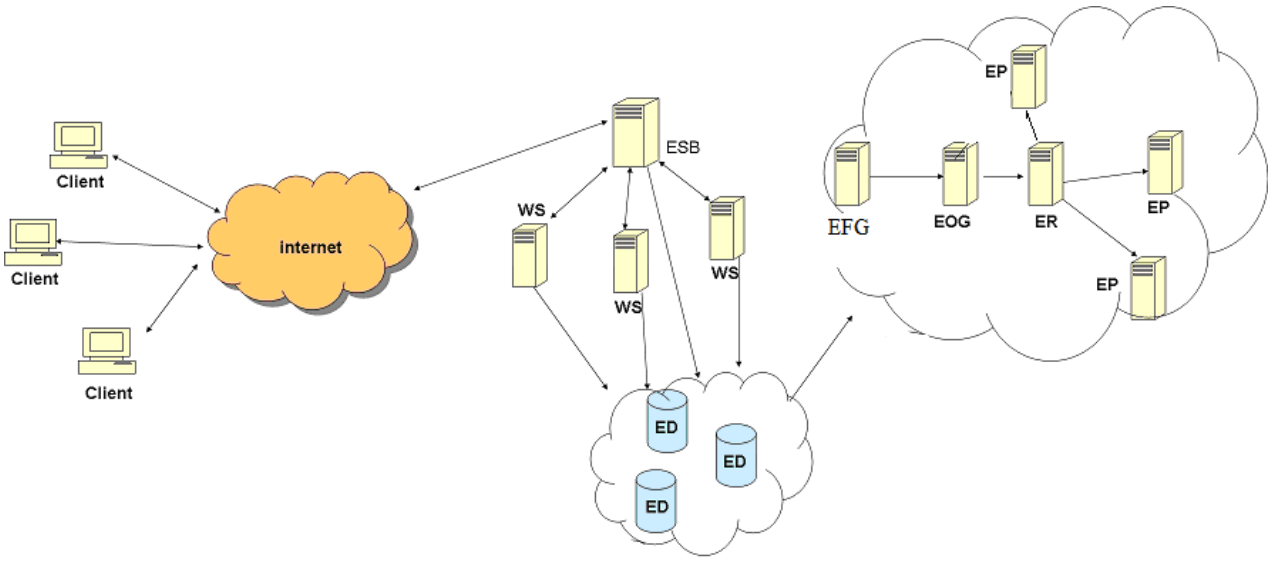

Figure 4-1: System Architecture

4.1 Enterprise Service Bus (ESB)

An Enterprise Service Bus is the main interface between the client and the web services. The client sends request for the web service to the ESB. ESB send the request to the proper WS. If the WS generates any response message of the request to the client that response is sent to the ESB at first and then the ESB sends the response to the client.

For example, let's assume that there is a web service provider who provides the current temperature of a place. If a client needs to know the temperature of a place it sends a request for the service to the ESB with the name of the place. ESB finds the service provider and sends the request to the WS. Then the WS replies with the temperature to the ESB. The ESB then forwards the temperature to the client.

In the context of our POMWS example the client will send the PO to the ESB. Then ESB will send the PO to the POMWS.

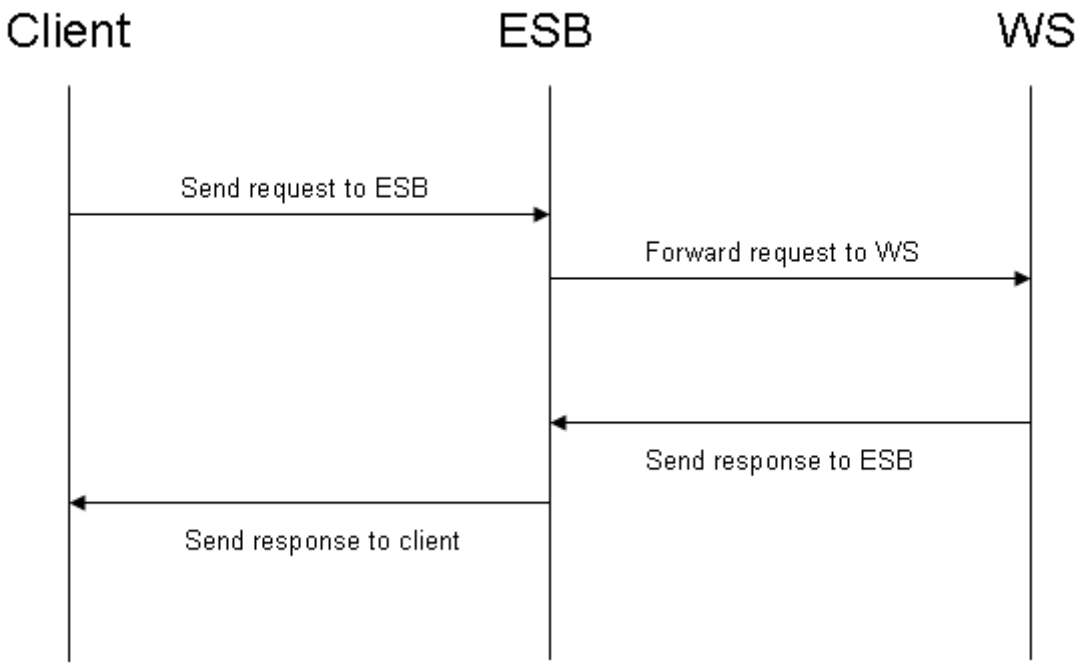

Figure 4-2: Message flow among Client, ESB and WS

The use of ESB has some advantages. If any WS is down then the client need not know about it. In this case ESB will re-direct the request to another WS which provides the same required service. This will improve availability of the service from the client's perspective. The use of ESB also shields the WS from malicious clients.

4.2 Event Database (ED)

In a WS network different events occur all the time. For example, a service request event occurs when a client requests a service, a service response event occurs when the WS provider sends a response of a service request to the service consumer. There are situations when faults can occur in the system. These faults are also events. All these events are reported to and stored in an Event Database (ED). The events are reported in a standard format. The format includes

information about event type, address of event source, importance of the event, event generation time and some other relevant information. The following figure shows the format of an event.

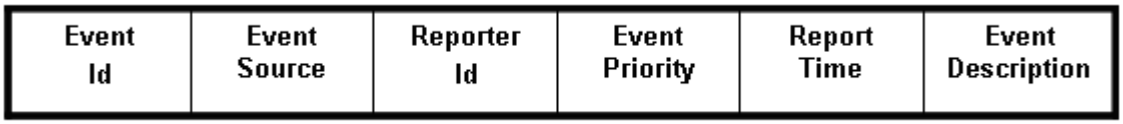

Figure 4-3: Event format

The event Id defines a globally unique id for the event. The event may be a service request event, service failure event, security event, status event, performance event etc.

The event source describes the source of the event. This can be the address of the event source or any other unique identifier of the event source.

The reporter id is the id of event detector that detects the event and reports the event to the event database. Event detectors can be the web service, client of a service, or the event processor itself.

The event priority field describes the priority of the event. The higher number signifies the more important event whereas the low numbers mean less important event. The report time is the time when the event occurred. The Event description is a general description of the event.

Figure 4-4 shows an example of a service request event. Here the event id is 101 that indicates a service request. Event source is client1. ESB is the reporter of the event. ESB reports the event to the event database. The priority of this event is 3. The priority has a scale from 5 to 1 an a higher number means higher priority. The event was reported at 1235375511706. The description of the event says that it is a service request.

101	client1	ESB	3	1235375511706	Service request

Figure 4-4: Example of an event

In the context of POMWS all the participating WS will send events to the event database. POMWS will also send events to the database. Some of the events reported by POMWS is defined below.

Service request: This event is reported when anyone makes a service request to a WS. For example, POMWS makes a service request to a participating WS.

Service timeout: This event is reported when a service requester does not get a response from the WS within a certain time. For example, POMWS does not get a response for a service request within a certain time.

Service retry: This event is reported when a service requester requests again for a service after a service timeout occurs. For example, POMWS re-sends a service request after a service time out occurred.

Service unavailable: This event is reported when a service requester finds a service unavailable after retrying. For example, POMWS does not get a response from the WS after retrying the service.

4.3 Event File Generator

The Event Stream Generator (EFG) collects the events from the event database and creates an XML file containing events. The Event Object Generator (EOG) generates event objects from the XML event files created by EFG.

The XML format of an event is shown in the following figure.

```
<event>
  <eventId>101</eventId>
  <eventSource>client1</eventSource>
  <reporterId>ESB</reporterId>
  <eventPriority>3</eventPriority>
  <reportTime>1235375511706</reportTime>
  <eventDescription>service request</eventDescription>
</event>
```

Figure 4-5: Example of an event in XML format

4.4 Event Object Generator (EOG)

EOG reads the XML event file created by EFG. Then EOG creates instance of events from the XML event elements. EOG sends these event instances to the Event Router (ER) over network connection.

4.5 Event Router (ER)

ER delivers the event instances to the Event Processors (EP). ER determines what events the EP can handle. To determine this ER distributes case scenarios to the EP. The case scenarios define different cases of patterns of events.

The case scenario files are in XML format. Each scenario has two main elements. It has a sequence of events and a list of actions. When the sequence of events is matched in the event

stream the actions will be executed. ER creates instances of scenarios from this XML scenario file and distributes the scenarios among the EPs.

The Event Router maintains connection with a number of Event Processors (EP). This connection can be a socket based connection or can be based on RMI. When an EP connects to the ER, the router sends case scenarios to the EP. Then ER sends the event instances for processing by EP.

The following figure shows a sample scenario file in XML format.

```
<?xml version="1.0"?>
<casescenarios>
        <scenario id ="1">
                <event>
                        <eventId>101</eventId><eventSource>InventoryWS</eventSource>
                        <reporterId>ESB</reporterId><eventPriority>2</eventPriority>
                        <reprotTime greaterThan="yes">1235375511706</reportTime>
                        <eventDescription>Service request</eventDescription>
                </event>
                <event>
                        <eventId>102</eventId><eventSource>InventoryWS</eventSource>
                        <reporterId>ESB</reporterId><eventPriority>3</eventPriority>
                        <reprotTime greaterThan="yes">1235375511706</reportTime>
                        <eventDescription>Service timeout</eventDescription>
                </event>
                <event>
                        <eventId>103</eventId><eventSource>InventoryWS</eventSource>
                        <reporterId>ESB</reporterId><eventPriority>3</eventPriority>
                        <reprotTime greaterThan="yes">1235375511706</reportTime>
                        <eventDescription>Service retry</eventDescription>
                </event>
                <event>
                        <eventId>104</eventId><eventSource>InventoryWS</eventSource>
                        <reporterId>ESB</reporterId><eventPriority>5</eventPriority>
                        <reprotTime greaterThan="yes">1235375511706</reportTime>
                        <eventDescription>Service unavailable</eventDescription>
                </event>
                <action><showMessage>Inventory service unavailable</showMessage></action>
        </scenario>
</casescenarios>
```

Figure 4-6: Example of a case scenario in XML format

4.6 Event Processor (EP)

The event processor (EP) processes the events the ER sends. Before an event processor finds a fault scenario in the events, it has to know the case scenarios. ER sends the case scenarios to the event processor. EP can work only on the scenarios that ER sent to them.

When EP is ready to process the events, ER sends the events to the EP. The Event Processor (EP) finds and matches patterns in these events. When any scenario is matched in the events, the EP can execute the actions described in the action part of the case scenario. This action can be

1. To send a report to the management application
2. To generate a new event and report to the ED.
3. To take any other action described in the action element.

For example in POMWS let us assume that the inventory WS is unavailable due to some software faults. In this case the following events should be reported.

4. POMWS reports a service request event where the requested WS is the inventory WS.
5. POMWS reports a service timeout event about the inventory WS.
6. POMWS reports a service retry event about the inventory WS.
7. POMWS reports a service unavailable event about the inventory WS.

From this sequence of events EP will detect the fault. Then EP will take action to make the inventory WS available. The action can be simply restarting the service.

CHAPTER 5
EVALUATION

5.1 Algorithm and Complexity

According to the proposed architecture events will be stored in the event database. The event file generator reads the events from the event database and writes them in files in XML format. The event object generator generates event objects from the files and sends them to the event router. The event router will send the event to the event processors that subscribe to the events. When event processors subscribe to event router for events they send the names of the required event sources.

```
Algorithm EventRouterThread

        getSourceNamesFromEventProcessor();
        storeSourceNamesInHashtable();
        sendCaseScenariosToEventProcessor();

        while(true)
        {
                getEventFromEventObjectGenerator();
                sendEventToEventProcessor();
        }
```

Figure 5-1: Pseudo code of event router

```
Algorithm EventProcessor

        sendSourceNamesFromEventProcessor();
        getCaseScenariosFromEventRouter();

        while(true)
        {
                getEventFromEventRouter();
                matchEventWithScenarios();
        }
```

Figure 5-3: Pseudo code of event processor

The event processor receives case scenarios from the event router and stores the scenarios in a vector. When an event arrives for checking event processor checks the scenario vector to match the event with an event in the case scenario. So the size of the scenario vector plays a role to determine the runtime. As multiple faults can occur in the system, the event processor has to check all the scenarios of the scenario vector. As a result, when the number of case scenarios increases the event processing time increases linearly. Processing of one event does not affect the processing time of the next event. For a constant number of case scenarios processing time of events has linear relation with the number of events. More events will take more time for processing.

If the number of events is n and the number of case scenarios is m then the complexity of the algorithm is *O(mn)*.

5.2 Experimental data

Experiments have been done to evaluate the performance of the system. The experimental data contains:

1 Event processing time for different numbers of fault scenarios.

2 Event processing time when Event Processors are not distributed.

3 Event processing time when Event Processors are distributed.

The proposed system has been implemented in the JAVA language. To evaluate the system eight simple WS and 50 clients are developed. The clients invoke these simple WS. During the runtime of the system different events are generated by the clients and the web services. These events are recorded in the event database (ED).

The simple WS are created using JAX-WS 2.0 with the Java SE 6 platform. JAX-WS is the Java API for XML Web Services. JAXWS version 2.0 is the center of the API stack for WS.

It includes Java Architecture for XML Binding (JAXB) 2.0 and SOAP with Attachments API for Java (SAAJ) 1.3.

The event database is implemented in MySQL Server 5.0. JDBC technology is used to connect to the database. JDBC technology is an API (included both in J2SE and J2EE) that provides connectivity with the databases. I used MySQL Connector/J 3.1 as the JDBC driver to implement the connection between the database and J2SE platform. Event File Generator, Event Object Generator, Event Router and Event Processors are implemented in Java programming language.

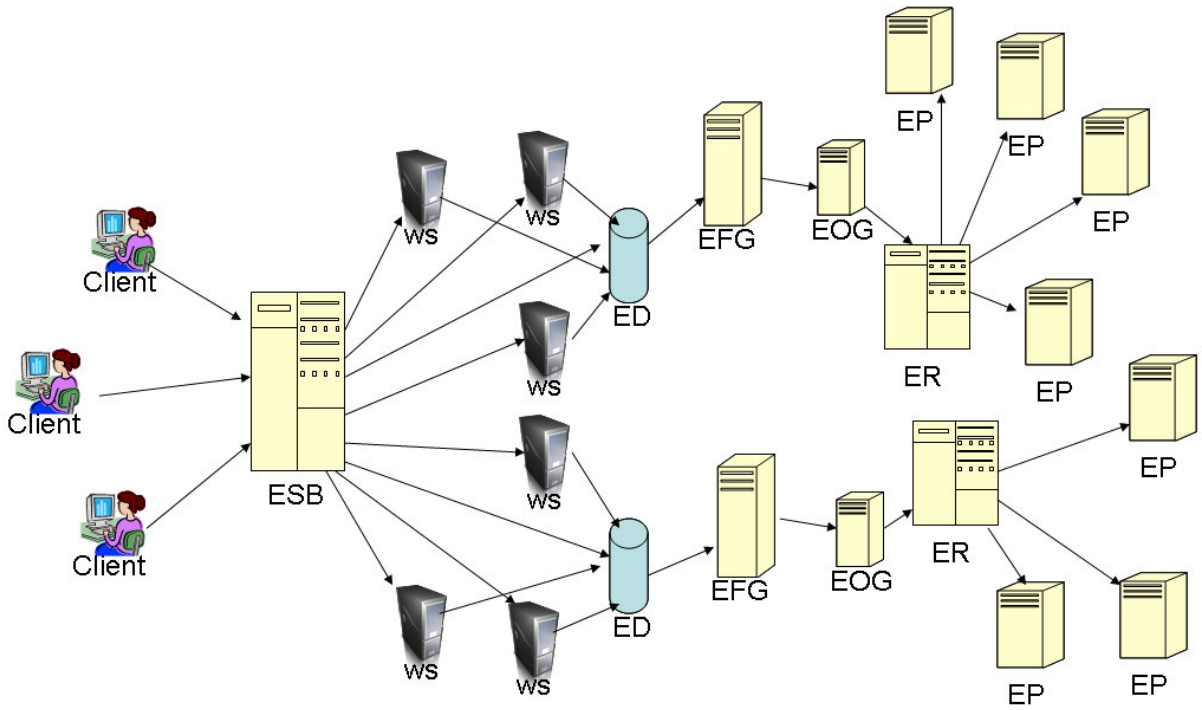

Figure 5-3: Experimental Setup

The setup was run in different large and small machines in Amazon EC2. Amazon Elastic Compute Cloud (Amazon EC2) is a web service to provide cloud computing facility to the developers. It provides complete control over the computing resources to the developer. Developers can configure small, medium, large or extra large instances. For compute intensive applications the instances can be High-CPU medium instance or High-CPU extra large instance.

In Amazon EC2 a standard large machine is a 64 bit platform. It has dual-Core-AMD Opteron processor 2218 HE of 3.07 GHz and 7.50 GB RAM. Operating system is Microsoft Windows Server 2003 R2 Datacenter x64 Edition with Service Pack 2. The storage size of an instance is 850 GB.

A standard small machine is a 32 bit platform. It has dual-Core-AMD Opteron processor 2218 HE of 2.60 GHz and 1.66 GB RAM. Operating system is Microsoft Windows Server 2003 R2 Datacenter Edition with Service Pack 2. The instance storage size is 160 GB.

5.2.1 Processing Time of Events for Different Number of Case Scenarios

In this experiment processing time was calculated for different number of case scenarios. This experiment was done in two large machines of amazon EC2. In the experiment the number of case scenarios was changed and two event processors were used. The events were stored in XML files and from these files the same event stream was sent for processing by event processors for different number of case scenarios.

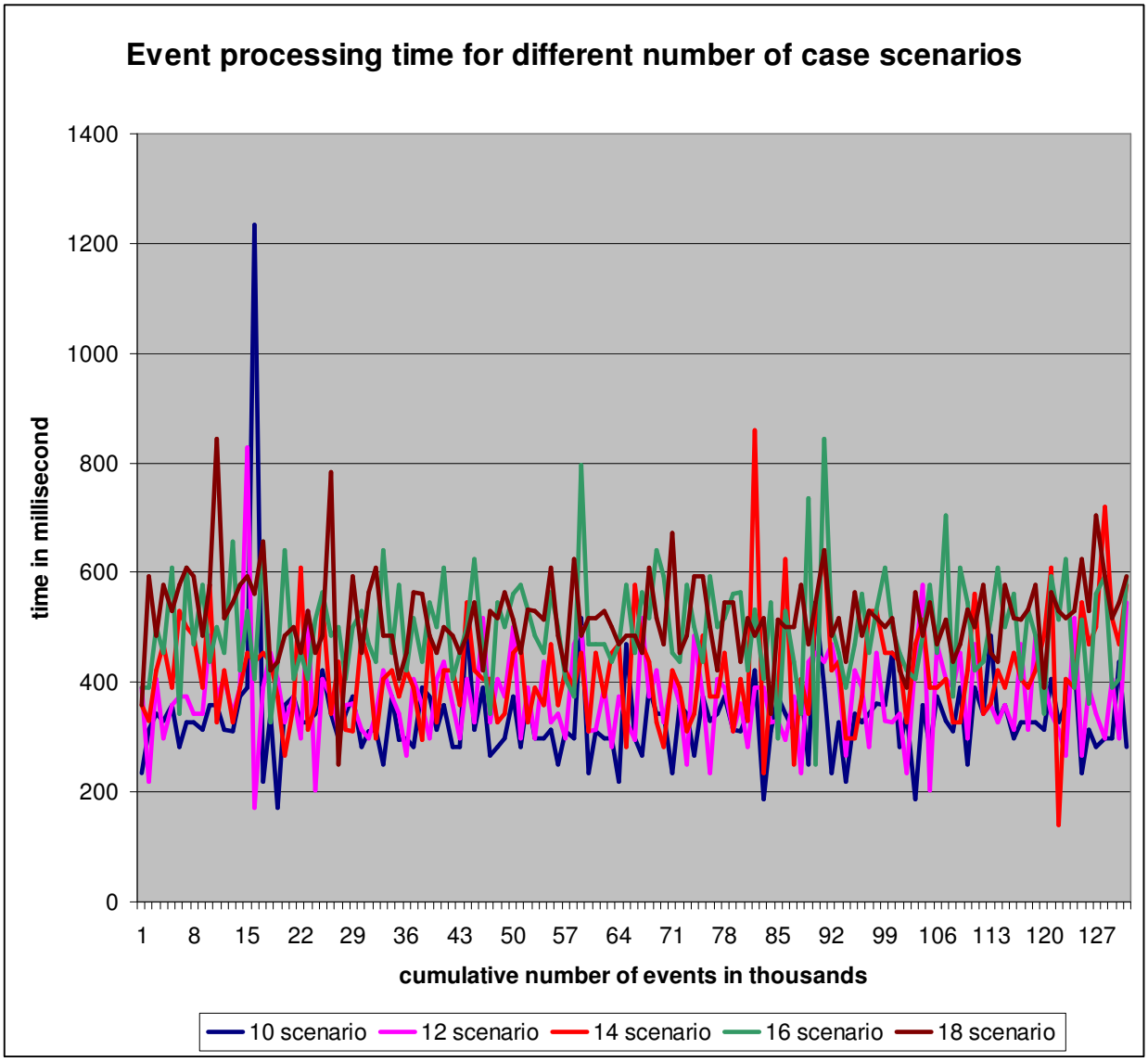

Figure 5-4: Event processing time for different number of case scenarios

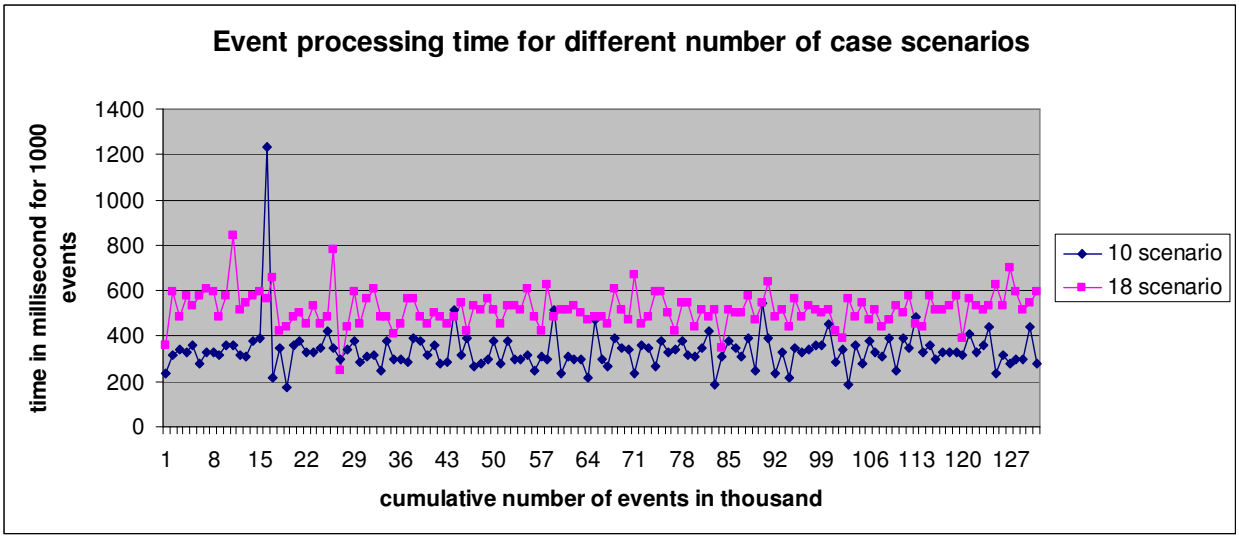

Figure 5-5: Event processing time for 10 and 18 case scenarios

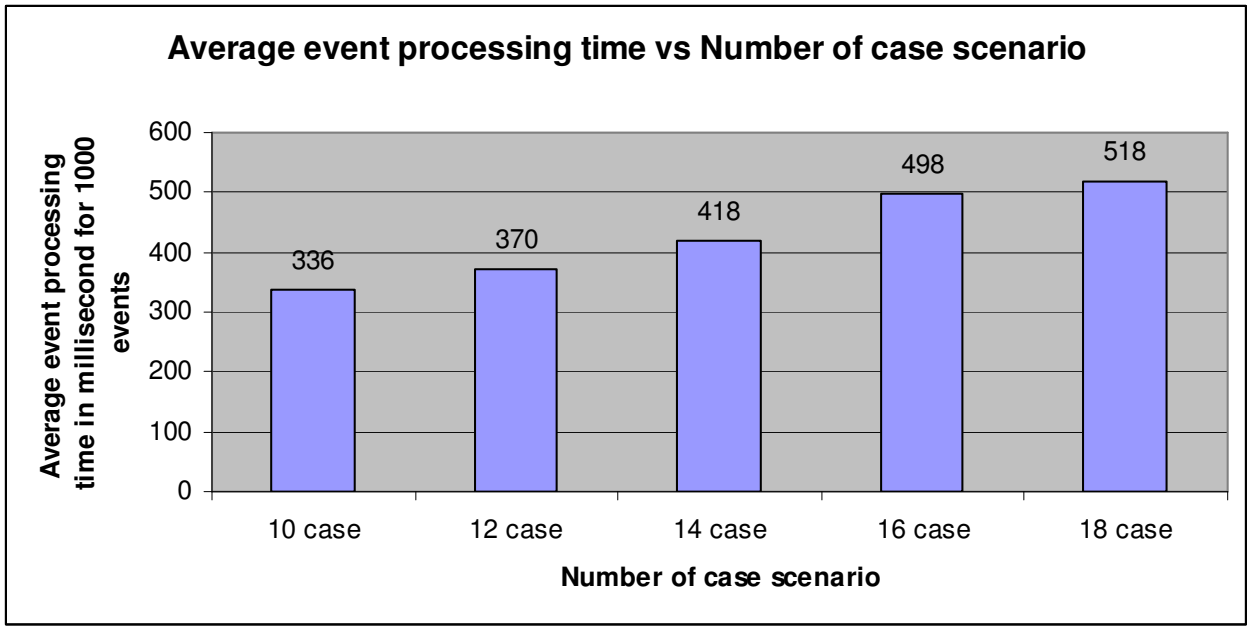

Figure 5-6: Average event processing time for different number of case scenarios

From the experimental data it can be seen that for a small number of case scenarios the processing time is less. When the number of case scenarios increases event processing time also increases. So when there are a lot of case scenarios the case scenarios should be distributed

among event processors so that each event processor has to process a small number of case scenarios. In this case more event processors need to be added in the system.

5.2.2 Processing Time of Events When Event Processors Are Not Distributed

In this experiment event processing time was calculated when the event processors and the event router were in a large machine. As EP and ER were in the same machine there was no network delay for sending events from ER to EP. In the experiment events were saved in files in XML format and sent to event processors i.e. same event stream was sent for different number of event processors.

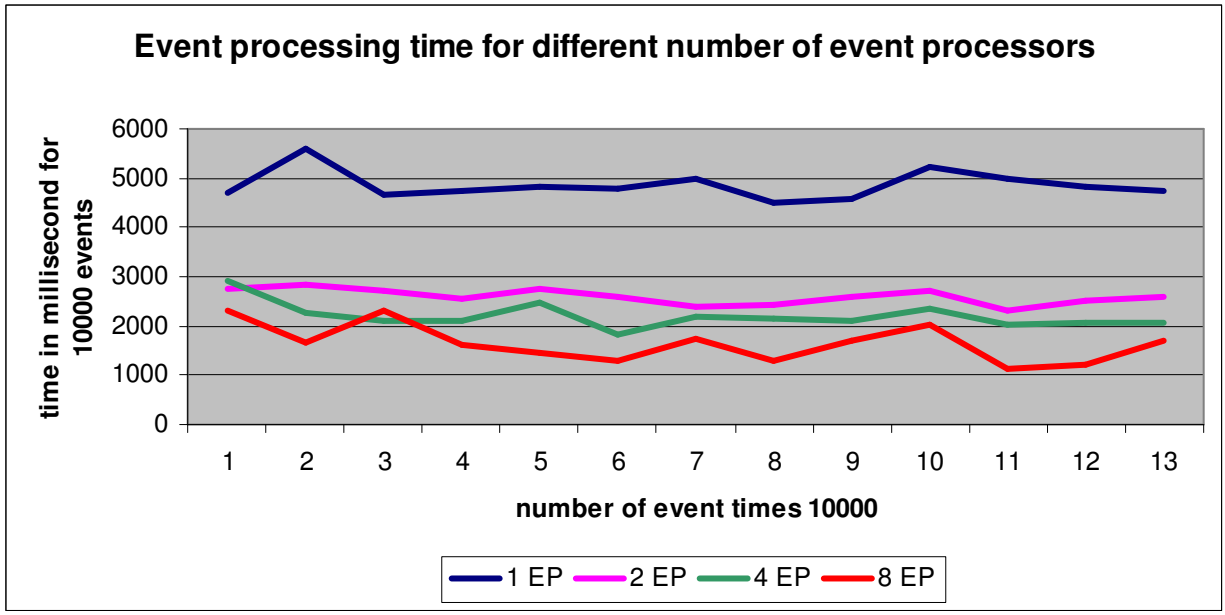

Figure 5-7: Event processing time per 10000 eventsfor different number of event processors

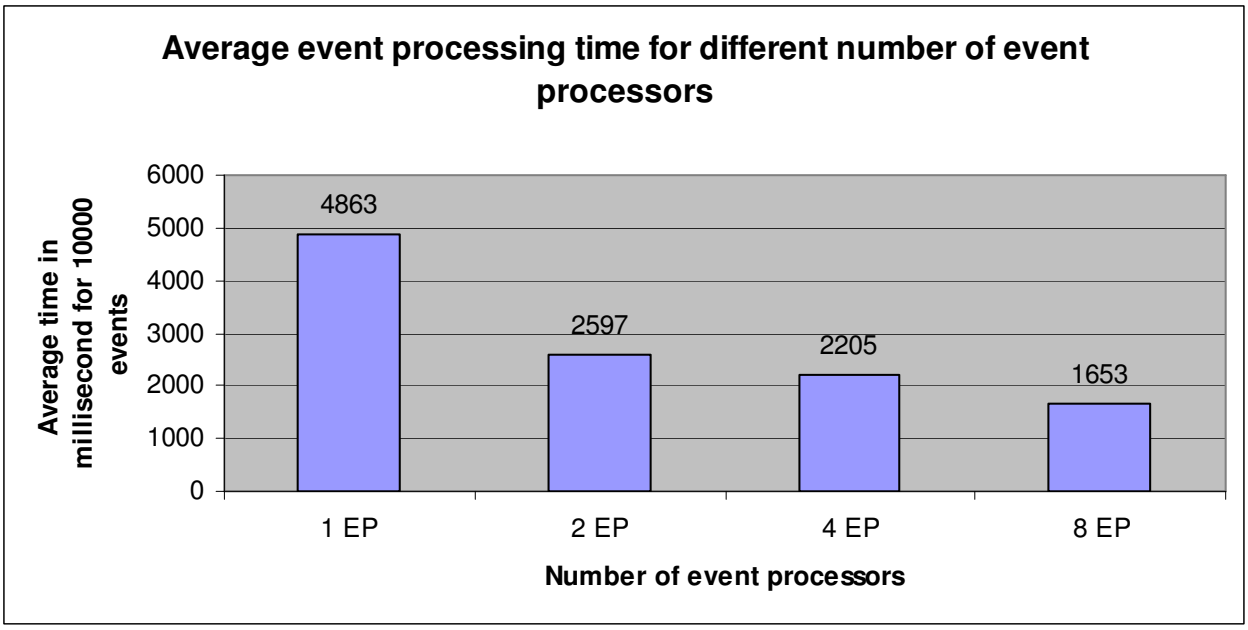

Figure 5-8: Average event processing time for different number of event processors

From the experimental data we can see that when the number of event processors increases event processing time decreases.

5.2.3 Processing Time of Events When Event Processors Are Distributed

In this experiment event processing time was calculated when the event processors and the event router were in two different large machines. As EP and ER were in the different machines there was network delay for sending events from ER to EP. In the experiment also events were saved in files in XML format and sent to event processors i.e. the same event stream was sent for different number of event processors.

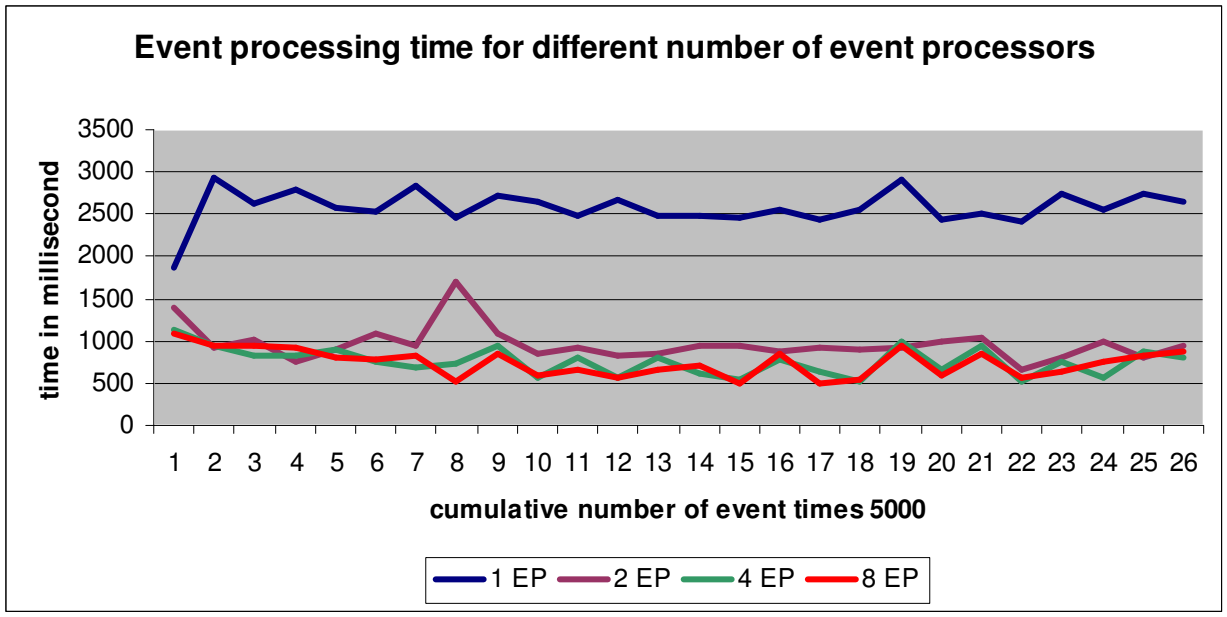

Figure 5-9: Event processing time per 5000 events for different number of event processors

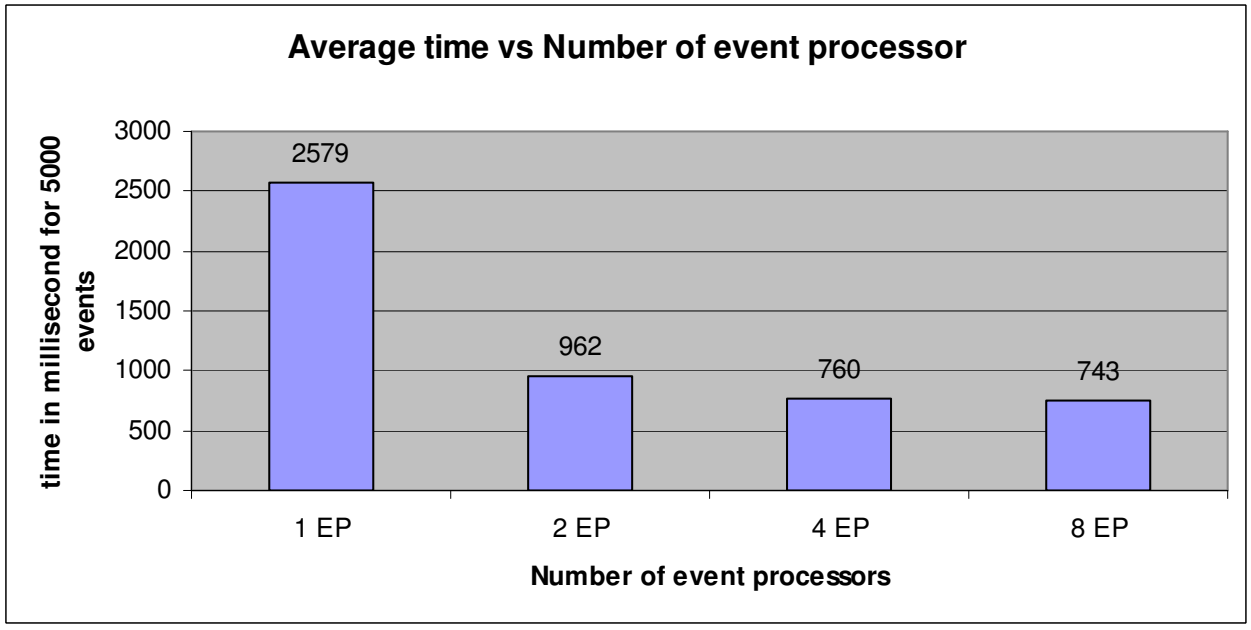

Figure 5-10: Average event processing time per 5000 events for different number of event processors

From the experimental data we can see that distributing event processing to different event processors improves the event processing time. When the number of event processors increases event processing time decreases. The processing time does not improve indefinitely. In the experimental data we can see that 4 and 8 event processors took nearly same amount of processing time. This is due to the low number of managed web services. In this experiment 8 web services were used. When there was only one event processor, the event processor processed events for all the 8 WS. When there were 4 EP, each EP processed events for 2 WS. In case of 8 EP each EP processed events for 1 WS. As 8 EP were underutilized, the processing time did not improve. So after adding certain number of event processors the processing time does not improve much because event processors do not become overloaded with the events. . Here the number of event processors depends on the number of WS in the system.

CHAPTER 6
CONCLUSIONS AND EXPECTED CONTRIBUTION

SOA is gaining its popularity in the service provider community. So the number of WS is increasing every day. To maintain desired quality of service WS require constant monitoring for faults and failures. Detection of faults and recovery from failure improves reliability and availability of the services. So fault management is a key issue in SOA. This research focuses on this key issue and presents an architecture for distributed fault management of WS.

In this work a new approach for fault management of web services has been presented. This approach uses distributed case based event processing. Using this approach a fault management system for SOAP based WS have been implemented and its performance has been tested. The event processing applications in the implemented system can process the events to detect faults to help monitoring and management of the services successfully. The components of the fault management system are distributed in the network. The system is scalable and reliable also.

The event processors receive the fault case scenarios and use them to detect faults in the system. From experimental data it can be seen that if the number of case scenarios increases then the processing time of events increases and if the number of event processors increases then the processing time of events decreases. So if there are a large number of case scenarios then these case scenarios can be distributed among the event processors. Then each event processor will have to work with a small number of case scenarios. This will improve the processing time of events.

The components of the fault management system are distributed over the network. Single large resource is not required for implementing the system. The system will not stop working for single point of failure.

The processing load can be made balanced among the event processors. All the event processors do not need to process all the events. One event processor will get only the events it has subscribed for. One event processor can look after the faults of some specific web services instead of monitoring and managing all web services.

The event processor needs to know the case scenarios related to the web services it is managing. The case scenarios are stored in XML formatted file. The creator of the case scenario file does not need to know all possible case scenarios of all the web services. He has to know only case scenarios related to the WS he is trying to manage. Anyone who has knowledge about creating XML files can create case scenario file and he does not need to have programming skills.

The system will be able to detect the faults that are described in the case scenario files. If any new fault occurs in the system and the fault management system can not detect it then the case scenario file has to be updated. The creator of the case scenario has to append the file with new scenarios.

The system is scalable since all the components are stand alone running processes. So in the system new event processors, event routers and event file generators, and event object generators can be easily included without stopping the running system.

The fault management system is reliable. As the event processors are distributed over the internet, single component failure will not stop running the management system. This improves the reliability of the fault management system.

There are some limitations in the implemented experimental setup. This work assumed that there will be no noise in reporting the events. But it is not always true in a real world scenario. Network connections are not always reliable. So due to network failure events can be lost.

In my work I assumed that the databases are reliable. If database is not reliable in the system then stored events can be lost. Storing the events in multiple databases can prevent event loss in case of database failure.

In this research work JMX technology was not used. In my future work I want to incorporate the use of JMX technology. JMX technology provides tools for building distributed, web based, modular and dynamic solutions for monitoring managing devices, applications, and service driven networks. JMX technology has the following advantages [56].

1. The JMX technology enables Java applications to be managed without heavy investment.
2. The JMX technology provides a standard way to manage Java applications, systems, and networks.
3. The JMX technology can be used for out-of-the-box management of the Java VM.
4. The JMX technology provides scalable, dynamic management architecture.
5. The JMX technology leverages existing standard Java technologies.
6. The JMX technology integrates with existing management solutions and emerging technologies.

In the future I also want to work on improving the analyzing capability of the event processing application. Then I want to use this approach to predict the future behavior of the services.

REFERENCES

[1] M. P. Papazoglou: “Web services: Principles and technology”. ISBN 9780321155559, Prentice Hall.

[2] M. Gudgin et al. (eds.), “SOAP 1.2 Part 1: Messaging Framework”, W3C Recommendation, June 2003, available at: http://www.w3.org/TR/2003/RECsoap12-part1-20030624/

[3] “The structure of a SOAP messge”, http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp?topic=/com.ibm.cics.ts31.doc/dfhws/concepts/soap/dfhws_message.htm

[4] T. Andrews et al. (eds.), “Business Process Execution Language for Web Services”, May 2003, available at http://www.ibm.com/developerworks/library/ws-bpel

[5] C. Pelz, “Web services Orchestration and Choreography”, Web Services Journal, July 2003.

[6] Workflow Management Coalition, “Terminology & Glossary”, Document number WFMC-TC-1011, February 1999.

[7] N. Kavantzas et al., “Web Services Choreography Description Language 1.0”, Editor’s draft, April 2004, available at http://lists.w3.org/Archives/Public/wwwarchive/2004Apr/att-0004/cdl_v1-editors-apr03-2004-pdf.pdf

[8] M. P. Papazoglou, “Extending the service oriented architecture”, Business Integration Journal, February 2005

[9] J. Murry, “Designing Manageable Applications”, Web Developer’s Journal, October 2002, available at http://www.webdevelopersjournal.com/articles/design_man_app/

[10] T. Mehta, “Adaptive Web Services Management Solutions”, Enterprise Networks and Servers, vol. 17, no. 5, May 2003, available at http://www.enterprisenetworksandservers.com/monthly/toc.php?35

[11] D. Kakadia et al., “Enterprise Management Systems: Architectures and standards”, Sun Microsystems, April 2002, available at: http://www.sun.com/blueprints/0402/ems1.pdf

[12] I. Sedukhin, “Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.0”, OASIS-Standard, March 2005, available at http://docs.oasisopen.org/wsdm/2004/12/wsdm-mows-1.0.pdf

[13] R. Robinson, “Understand Enterprise Service Bus Scenarios and Solutions in Service-Oriented Architecture”, IBM Developer Works, June 2004, available at http://www-106.ibm.com/developerworks/library/ws-esbscen/

[14] A. Candadai, “A Dynamic Implementation Framework for SOA-based Applications”, Web Logic Developers Journal: WLDJ, September/October 2004.

[15] K. Channabasavaiah, K. Holley, E. M. Tuggle, Jr. “Migrating to a Service Oriented Architecture”, IBM Developers Works, December 2003, available at http://www-106.ibm.com/developerworks/library/ws-migratesoa/

[16] D. Chappell, Enterprise Service Bus, O’Reilley, 2004

[17] M Endrei et al., “Patterns: Service-Oriented Architecture and Web Services”, IBM Redbooks SG24-6303-00, April 2004.

[18] JMX technology home page is available at: http://java.sun.com/javase/technologies/core/mntrmgmt/javamanagement/

[19] MUWS documentation is available at: http://docs.oasisopen.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf

[20] MOWS documentation is available at: http://www.oasisopen.org/committees/download.php/20574/wsdm-mows-1.1-spec-os-01.pdf

[21] CIM specification is available at: http://www.dmtf.org/standards/cim/cim_spec_v22

[22] SNMP specification is available at:http://www.sei.cmu.edu/str/indexes/references/SNMPv2_Specs.html

[23] WS-Eventing specification available at: http://www.w3.org/Submission/WS-Eventing/

[24] R. López de Mántaras and E. Plaza, "Case-Based Reasoning: An Overview". AI Communications, ISSN 0921-7126 (Print) 1875-8452 (Online), Issue Vol. 10, Nr 1, 1997 Pages 21-29.

[25] R. Schank and R. Abelson (eds.), "Scripts, plans, goals and understanding: an inquiry into human knowledge structures.", Lawrence Erlbaum Associates, Hillsdale, NJ, 1977 248 p.

[26] R. Schank, Dynamic Memory: "A Theory of Reminding and Learning in Computers and People", Cambridge University Press, New York, 1982.

[27] A. Aamodt and E. Plaza, "Case-based reasoning: foundational issues, methodological variations and System approaches". AI Communications, Vol. 7 Nr. 1, March 1994, pages 39-59.

[28] J. Carbonell, "Learning by analogy: Formulating and generalizing plans from past experience. In (Michalski, Carbonel and Mitchell, eds): Machine Learning: An Artificial Intelligence Approach", Tioga, Palo Alto.

[29] J. Kolodner, "Reconstructive memory, a computer model". Cognitive Science Volume 7, Issue 4, October-December 1983, Pages 281-328.

[30] E.L. Rissland, "Examples in legal reasoning: legal hypotheticals". Proceedings IJCAI'83, Karlsruhe.

[31] D. B. Leake, “CBR in Context: The Present and Future”. Chapter from “Case based reasoning: experiences, lessons, and future directions”, AAAI Press/MIT Press, 1996.

[32] B. Ross, “Remindings and their e_ects in learning a cognitive skill”. Cognitive Psychology, vol. 16 nr. 3, Jul 1984, pages 371-416.

[33] P. Pirolli, and J. Anderson, “The role of learning from examples in the acquisition of recursive programming skills”. Canadian Journal of Psychology vol. 39, 1985, pages 240-272.

[34] J. Faries and K. Schlossberg, The effect of similarity on memory for prior problems. In Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society,1994, pages 278-282.

[35] J. Lancaster and J. Kolodner, “Problem solving in a natural task as a function of experience”. In Proceedings of the Ninth Annual Conference of the Cognitive Science Society, 1987, pages 727-736.

[36] H. Schmidt, G. Norman, and H. Boshuizen, “A cognitive perspective on medical expertise: Theory and implications”. Academic Medicine vol. 65 issue 10, 1990, pages 611-621.

[37] S. Read and I, Cesa, “This reminds me of the time when . . . : Expectation failures in reminding and explanation”. Journal of Experimental Social Psychology vol. 27, 1991, pages 1-25.

[38] G. Klein, and R. Calderwood, “How do people use analogues to make decisions?” In Kolodner, J., ed., Proceedings of the DARPA Case-Based Reasoning Workshop, 1988, pages 209-223.

[39] G. Klein, and R. Calderwood, “Decision models: Some lessons from the _eld”. IEEE Transactions on Systems, Man, and Cybernetics 1989, vol. 21(5) pages 1018-1026.

[40] J. Kolodner, “Case-Based Reasoning”. San Mateo, CA: Morgan Kaufmann, 1993.

[41] C. Owens, “Indexing and retrieving abstract planning knowledge”. Ph.D. Dissertation, Yale University, 1991.

[42] E. Rissland, J. Kolodner, and D. Waltz, Case-based reasoning. In Hammond, K., ed., Proceedings of the DARPA Case-Based Reasoning Workshop, 1-13, 1989.

[43] http://java.sun.com/developer/technicalArticles/WebServices/restful/

[44] R. Monson-Haefel, J2EE Web Services, Addison-Wesley, 2004.

[45] http://www.w3schools.com/soap/soap_example.asp

[46] http://www.w3.org/TR/wsdl

[47] M. P. Papazoglou, W. J. van den Heuvel, “Web services Management: A survey”, IEEE Internet Computing, November/December 2005.

[48] D. Ardanga et al. “Faults and recovery actions for self healing web serives, available at ftp://ftp.elet.polimi.it/users/Barbara.Pernici/ws-diamond/Faults-Recovery-Actions-Polimi-nov05.pdf

[49] W. He, “Recovery in Web service applications”, IEEE International Conference on e-Technology, e-Commerce and e-Service, 2004. EEE '04. 2004, 28-31 March 2004 Page(s):25 – 28

[50] M. Mansouri-Samani, M. Sloman, “Monitoring distributed systems” Network, IEEE Volume 7, Issue 6, Nov 1993 Page(s):20 – 30 Digital Object Identifier 10.1109/65.244791

[51] A. Benharref , R. Glitho and R. Dssouli, “Mobile Agents for Testing Web Services in Next Generation Networks”. Available at http://www.springerlink.com/content/y67740g27mjk5x87/

[52] E. Akbas, “System independent and distributed fault management system” Eighth IEEE International Symposium on Computers and Communication, 2003. (ISCC 2003). Proceedings.

Publication Date: 30 June-3 July 2003 On page(s): 1359- 1363 vol.2

http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1214302

[53] P. Kumar , “Web services and IT management”, Queue - Volume 3 , Issue 6 (July/August 2005) Pages: 44 - 49 , available at: http://portal.acm.org/citation.cfm?doid=1080862.1080876

[54] D. Luckham, “ The power of events, An introduction to complex event processing in distributed enterprise systems”. Chapter 5, Section 5.1, page 88, ISBN 978-0-201-72789-0.

[55] IEEE Standard Glossary of Software Engineering Terminology. lEEE Std 610.121990 (Revision and reddgnation of IEEEstd7921983) available at: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00159342

[56] “Why use the JMX technology”, Available at: http://72.5.124.55/docs/books/tutorial/jmx/overview/why.html